Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/KR05/000668

International filing date: 09 March 2005 (09.03.2005)

Document type: Certified copy of priority document

Document details: Country/Office: KR
Number: 10-2004-0093309
Filing date: 08 November 2004 (08.11.2004)

Date of receipt at the International Bureau: 30 June 2005 (30.06.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

대한민국 특허청
KOREAN INTELLECTUAL PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Intellectual Property Office

출 원 번 호 : 특허출원 2004년 제 0093309 호
Application Number 10-2004-0093309

출 원 일 자 : 2004년 11월 08일
Date of Application NOV 08, 2004

출 원 인 : 양세양
Applicant(s) YANG, Sei Yang

2005 년 06 월 09 일

특 허 청
COMMISSIONER

【서지사항】

【서류명】 특허출원서

【권리구분】 특허

【수신처】 특허청장

【참조번호】 0001

【제출일자】 2004.11.08

【발명의 국문명칭】 검증 성능을 높이는 시뮬레이션 기반의 검증 장치 및 이를 이용한 시뮬레이션 방법

【발명의 영문명칭】 Simulation-based Verification Apparatus Achieving High Verification Performance, and the Simulation Method Using the Same

【출원인】

【성명】 양세양

【출원인코드】 4-1998-037998-4

【발명자】

【성명】 양세양

【출원인코드】 4-1998-037998-4

【우선권 주장】

【출원국명】 KR

【출원종류】 특허

【출원번호】 10-2004-0017476

【출원일자】 2004.03.06

【증명서류】 미첨부

【취지】 특허법 제42조의 규정에 의하여 위와 같이 출원합니다. 출원인 양세양 (인)

【수수료】

【기본출원료】	0	면	38,000 원
【가산출원료】	38	면	38,000 원
【우선권주장료】	1	건	20,000 원
【심사청구료】	0	항	0 원
【합계】	96,000 원		
【감면사유】	개인(70%감면)		
【감면후 수수료】	42,800 원		
【첨부서류】	1.요약서 · 명세서(도면)_2통		

【요약서】

【요약】

본 발명은 설계된 매우 복잡한 디지탈 시스템의 설계 검증을 위한 검증 장치와 이를 이용한 효과적인 시뮬레이션 방법에 관한 것이다.

본 발명에서는 임의의 컴퓨터에서 수행되어지는 본 발명의 검증 소프트웨어로 하여금 설계 코드에 부가적인 코드를 부가하여 시뮬레이션을 수행하게 한다. 시뮬레이션 수행은 1차 시뮬레이션과 2차 시뮬레이션으로 나누어지며 2차 시뮬레이션은 1 이상의 컴퓨터에서 수행되는 1 회 이상의 시뮬레이션을 1 이상의 시뮬레이터를 이용하여 병렬적인 수행을 가능하게 함으로서 전체 검증 시간의 대폭적인 단축을 가능하게 한다.

【대표도】

도 1

【명세서】

【발명의 명칭】

검증 성능을 높이는 시뮬레이션 기반의 검증 장치 및 이를 이용한 시뮬레이션 방법 {Simulation-based Verification Apparatus Achieving High Verification Performance, and the Simulation Method Using the Same}

【도면의 간단한 설명】

<1> 도1 은 본 발명에 관한 설계 검증 장치의 일 예를 개략적으로 도시한 도면.

<2> 도2 는 본 발명에 관한 설계 검증 장치의 또 다른 일 예를 개략적으로 도시한 도면.

<3> 도3 은 본 발명에서 제안하는 방법으로 시뮬레이션을 수행하는 과정을 개략적으로 도시한 도면.

<4> 도4 는 본 발명에서 제안하는 1차 시뮬레이션과 1차 이후의 시뮬레이션을 통하여 설계 오류를 발견하고 수정하는 과정을 개략적으로 도시한 도면.

<5> 도5(a) 은 도1 과 같은 장치를 이용한 설계 검증을 수행하는 과정을 개략적으로 도시한 순서도를 도시한 도면.

<6> 도5(b) 는 도2 와 같은 장치를 이용한 설계 검증을 수행하는 과정을 개략적으로 도시한 순서도를 도시한 도면.

<7> 도6(a) 는 도2 와 같은 장치를 이용한 설계 검증에서 2차 이후의 시뮬레이션을 병렬적으로 수행하는 과정을 개략적으로 도시한 도면.

<8> 도6(b) 는 도2 와 같은 장치를 이용한 설계 검증에서 2차 이후의 시뮬레이션을 병렬적으로 수행하는 또 다른 과정을 개략적으로 도시한 도면.

<9> 도6(b) 는 도2 와 같은 장치를 이용한 설계 검증에서 2차 이후의 시뮬레이션을 병렬적으로 수행하는 또 다른 과정을 개략적으로 도시한 도면.

<10> 〈도면의 주요부분에 대한 부호의 설명〉

<11> 32 : 검증 소프트웨어 34 : 시뮬레이터

<12> 35 : 컴퓨터

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<13> 본 발명은 설계된 수백만 게이트급 이상의 디지탈 시스템의 설계를 검증하는 기술에 관한 것으로, 설계된 수백만 게이트급 이상의 디지탈 시스템을 시뮬레이션을 통하여 검증하고자 하는 경우에 검증의 성능을 증가시키는 검증 장치 및 이를 이용한 시뮬레이션 방법에 관한 것이다.

<14> 최근에 집적회로의 설계 및 반도체 공정기술이 급격하게 발달함에 따라 디지탈 회로 설계의 규모가 최소 수백만 게이트급에서 수천만 게이트급까지 커짐은 물론 그 구성이 극히 복잡해지고 있는 추세이고, 이와 같은 추세는 계속적으로 확대되고 있는 추세로 가까운 미래에 일억 게이트급 이상의 설계도 예상되고 있다. 그러나, 시장에서의 경쟁은 더욱 더 치열해지므로 빠른 시간 내에 우수한 제품을 개

발하여야만 함으로 빠른 시간 내에 자동화된 방법으로 설계된 회로를 효율적으로 설계 검증하기 위한 효과적인 방법의 필요성이 더욱 커지고 있다.

<15> 지금까지는 설계된 디지탈 회로를 설계 검증하기 위하여서 하드웨어 기술언어(Hardware Description Language, 앞으로 이를 HDL로 약칭함)를 이용한 설계 초기에는 소프트웨어적 접근법인 HDL 시뮬레이터들(예로 Verilog 시뮬레이터, VHDL 시뮬레이터, 또는 SystemC 시뮬레이터 등)이 주로 사용되어지고 있다. 이와 같은 시뮬레이터는 설계 검증 대상회로를 소프트웨어적으로 모델링한 순차적인 인스트럭션 시퀀스로 구성된 소프트웨어 코드를 컴퓨터 상에서 순차적으로 수행하여야 함으로 상기 수백만 게이트급 이상의 설계에 대해서는 시뮬레이션 성능의 저하가 설계 대상의 크기에 비례하여 발생하는 것이 문제가 되고 있다. 일 예로 1000만 게이트급 이상의 설계를 HDL 시뮬레이터로 시뮬레이션하는 경우에 현존하는 제일 빠른 프로세서를 장착한 컴퓨터에서 해당 HDL 시뮬레이터로 설계를 시뮬레이션하는 경우에 시뮬레이션 속도는 레지스터전송수준(Register Transfer Level, 앞으로 이를 RTL로 약칭함)으로 하는 경우에 10 cycles/sec를 넘기기 어려우며, 게이트 수준에서 시뮬레이션을 진행하면 1-2 cycles/sec를 넘기기가 어려운 것이 매우 일반적이다. 그러나 해당 설계를 검증하고자 필요한 시뮬레이션 사이클은 최소 수백만 사이클에서부터 최대 수십억 사이클이 필요함으로 전체 시뮬레이션 시간은 상상을 초월하게 된다.

<16> 이와 같은 장시간의 검증 시간을 단축하기 위하여 현재에 사용되는 기술들은 다음과 같은 것들이 있는데, 첫째는 하드웨어 기반의 검증 시스템(예로 시뮬레이션

가속기, 하드웨어 에뮬레이터, FPGA 프로토타이핑 시스템등)을 사용하거나, 둘째는 1 이상의 컴퓨터(예로 100대의 워크스테이션)들 가각에 HDL 시뮬레이터를 인스톨하고 이를 고속의 네트워크를 통하여 연결한 시뮬레이션팜(simulation farm)을 사용하는 것이다. 그러나 하드웨어 기반의 검증 시스템을 사용하는 것은 설계 초기에는 적용이 불가능하고, 합성이나 컴파일 과정이 HDL 시뮬레이터를 사용하는 것보다 오래 걸리며, 사용하기가 HDL 시뮬레이터에 비하여 매우 어렵고, 시스템의 구입 비용과 유지/보수 비용이 매우 클 뿐만 아니라, 무엇보다도 설계자나 검증엔지니어들이 HDL 시뮬레이터에 대한 선호도가 이들 하드웨어 기반의 검증 시스템에 비하여 매우 높고, HDL 시뮬레이터로는 아무 문제 없이 수행이 되는 설계 코드들이 하드웨어 기반의 검증 시스템에서는 수행되지 않는 경우가 많아서 이들 하드웨어 기반의 검증 시스템들은 제한적인 상황과 제한적인 사용자들에서만 사용되고 있다. 또한 시뮬레이션팜을 이용하여 시뮬레이션의 성능을 향상시키는 것은 시뮬레이션을 위한 설계 코드나 테스트벤치가 2이상의 경우에만 가능할뿐만 아니라, 테스트벤치들이 여러 개인 경우라고 하더라도 이들 중에서 제일 수행시간을 요하는 테스트벤치에 의하여 전체 시뮬레이션 시간이 결정되는 문제점(예를 든다면 특정 테스트벤치에 의한 시뮬레이션 시간이 일주일을 요하는 경우에는 시뮬레이션 팜(simulation farm)을 이용하더라도 일주일의 시뮬레이션 시간은 더 이상 단축이 안됨)이 있다.

【발명이 이루고자 하는 기술적 과제】

<17> 따라서, 본 발명의 목적은 초대규모급 디지털시스템 설계에 대한 검증을 위한 시뮬레이션의 성능 향상을 효율적으로 가능하게 하는 시뮬레이션 기반의 설계

검증 장치 및 이를 이용한 설계 검증 방법을 제공함에 있다. 특히 시뮬레이션을 수행한 후에 설계 오류에 대한 디버깅을 수행하기 위하여 설계 코드에 존재하는 시그널들이나 변수들에 대한 가시도(visibility)가 요구되는데, 문제는 설계 오류의 정확한 위치를 알아내기 위해서 시물레이션 수행 전에 구체적으로 어느 특정 시그널들이나 변수들에 대한 가시성이 필요한지를 예측하기 어렵다는 것이다. 따라서 시뮬레이션을 수행할 때에 처음부터 설계 코드에 존재하는 모든 시그널들과 변수들에 대하여 탐침이 가능하게 이들 모두를 덤프(dump) 대상으로 선정한 다음에 시뮬레이션을 수행한다. 그러나 설계 코드에 존재하는 모든 시그널들과 변수들을 덤프하면서 시뮬레이션을 수행하는 경우에는, 덤프를 전혀 하지 않고 시뮬레이션을 수행하는 것과 비교하여 시뮬레이션 수행시간이 대략 2배에서 많게는 10배 이상 길어지게 된다. 본 발명에서는 설계 코드에서 버그의 위치를 정확히 파악하기 위하여 설계 코드에 존재하는 모든 시그널들과 변수들에 대하여 시뮬레이션 처음부터 덤프를 수행하는 기존의 방법과는 다르게 시뮬레이션 수행시간을 늘리지 않고서도 설계 코드에서 버그의 위치를 찾을 수 있도록 하는 자동화된 방법과 이를 위한 검증 장치를 제공함을 또 다른 목적으로 한다.

【발명의 구성】

<18> 상기 목적들을 달성하기 위하여, 본 발명에 따른 설계 검증 장치는 검증 소프트웨어와 반도체설계용 시뮬레이터가 인스톨된 1이상의 컴퓨터로 구성된다. 검증 소프트웨어는 컴퓨터에서 실행되며, 만일 상기 설계 검증 장치에 2이상의 컴퓨터들이 있는 경우에는 이들 2이상의 컴퓨터는 네트워크로 연결되어져서 컴퓨터들 간에

파일들의 이동을 가능하게 한다.

<19> 본 발명에서 제안되는 검증 장치와 검증 방법은 하드웨어 설계 코드 자체를 검증하는 함수적 검증(functional verification)에 사용될 수 있을 뿐만 아니라, 설계코드를 합성한 게이트수준의 네트리스트를 이용한 게이트수준의 검증에서도 사용될 수 있고, 또는 배치(placement) 및 배선(routing)이 되고 추출된 타이밍정보를 게이트수준의 네트리스트에 첨부시켜(back-annotated) 수행하는 타이밍 검증에서도 사용될 수 있다. 그러나 앞으로의 설명은 설계 코드 자체를 검증하는 함수적 검증에 대하여 하기로 하며, 게이트수준 검증이나 타이밍 검증에 대해서도 같은 방법을 적용할 수 있음으로 구체적 설명은 생략하기로 한다.

<20> 검증 소프트웨어는 설계 코드를 읽은 후에 여기에다 추가적인 코드를 부가한다. 부가된 코드는 기본적으로 설계 코드로써 시뮬레이션을 수행하는 과정에서 일정 간격으로 시뮬레이션의 상태를 저장하는 역할을 수행하고, 또한 추후에 사용자의 요구에 따라서 저장된 시뮬레이션의 상태로부터 다시 시뮬레이션을 재개하게 한다. 시뮬레이션의 상태란 시뮬레이터라는 소프트웨어 프로그램이 수행되는 과정에서의 특정 시점에서 시뮬레이터의 모든 동적 정보를 일컫는 말이다. 이는 멀티프로그래밍 환경에서 임의의 프로그램(혹은 프로세스 내지는 쓰레드)이 잠시 수행을 멈추는 상태(wait)로 갔다가 나중에 다시 재개(resume)하기 위해서 저장이 되어져야 하는 프로그램의 상태 정보와 유사하다. 또한 이와 같은 저장된 시뮬레이션의 상태로부터 다시 시물레이션을 재개하는 경우에는 사용자의 의도에 따라서 설계 코드에 존재하는 모든 시그널들과 변수들에 대한 탐침을 수행하기 위한 덤프를 병행하거나

혹은 설계 코드에 존재하는 특정 시그널들이나 변수들에 대한 탐침을 수행하기 위한 덤프를 병행한다. 반도체설계용 시뮬레이터로 HDL 시뮬레이터(Verilog 시뮬레이터나 VHDL 시뮬레이터)를 사용하는 경우에 시뮬레이션의 상태를 저장하는 방법의 한 예로서 HDL 시뮬레이터의 save(NC-Verilog, Verilog-XL, VCS의 경우)나 혹은 checkpoint(Modelsim의 경우)라는 명령을 사용하고, 시뮬레이션을 특정 시간에서부터 재개하는 방법의 한 예로서 HDL 시뮬레이터의 restore(ModelSim) 혹은 restart(NC-Verilog, Verilog-XL, VCS)와 같은 명령을 사용할 수 있다. 이와 같은 시뮬레이션 방법이 검증의 성능 향상 및 효율성을 증대시킬 수 있는 이유는 다음과 같다. 이미 언급된 대로 시뮬레이션을 통하여 검증을 수행하는 과정에서는 설계 오류를 발견하고 이를 수행하는 과정에서 반드시 설계 코드에 존재하는 특정 시그널들이나 변수들의 값들을 특정 시간대에서 알 수 있도록 하는 탐침(probing)하는 과정이 항상 필요하다. 그러나 문제는 이와 같은 설계 코드에서 설계 오류를 발견하고 이를 수정하기 위해서 탐침이 필요한 특정 시그널들이나 변수들이 어떤 것인지를 시뮬레이션 수행 전에 정확히 예측할 수가 없을 뿐만 아니라, 이들 탐침 대상의 탐침 시점이 언제 필요한지도 시뮬레이션 수행 전에 정확히 예측할 수가 없다. 따라서 시뮬레이션을 1차 적으로 수행한 후에 이 1차 시뮬레이션 결과를 바탕으로 설계 오류의 위치를 파악하기 위하여 필요한 특정 시뮬레이션 시점에서의 특정 시그널들이나 변수들을 탐침 대상으로 선정한 후에 2차 시뮬레이션을 시뮬레이션 시간 0에서부터 1차 시뮬레이션 종료시점까지 진행하면서 탐침 대상이 된 시그널들이나 변수들을 특정 시간대에서 덤프를 수행하게 된다. 이와 같이 2차 시뮬레이션 과정

에서도 설계 오류의 위치를 파악하지 못하면, 새로운 시그널들이나 변수들을 탐침 대상으로 선정하고 시뮬레이션을 다시 시뮬레이션 시간 0에서부터 반복하게 되며, 이와 같은 과정을 설계 오류의 위치가 발견되기까지 수 차례 반복하는 과정이 필요하다. 그러나 이와 같은 반복적인 시뮬레이션을 시뮬레이션 시간 0에서부터 2회이상 반복하게 됨으로서 전체 검증 시간은 크게 늘어나게 된다. 만일 이와같은 반복적인 시뮬레이션을 피하고자 하는 경우에는 제일 처음 수행하는 시뮬레이션 수행시에 설계 코드에 존재하는 모든 시그널들과 변수들을 탐침 대상으로 설정하고 시뮬레이션 전 과정에 걸쳐서 이들을 덤프하면서 진행하여야 한다. 그러나 이와 같이 설계 코드에 존재하는 모든 시그널들과 변수들을 덤프하면서 시뮬레이션을 진행하게 되면 덤프를 진행하지 않고 시뮬레이션을 진행하는 것에 비하여 시뮬레이션 시간이 대략 2배에서부터 최대 10배 이상 증가하게 되어 이 또한 전체 검증 시간을 크게 늘리게 된다. 뿐만 아니라 이와 같이 설계 코드에 존재하는 모든 시그널들과 변수들을 전체 시뮬레이션 구간에 걸쳐서 덤프하게 되면 덤프되는 데이터의 크기가 수십 기가바이트에서부터 수백 기가바이트 이상 증가하게 된다. 이와 같은 큰 용량의 데이터를 저장하기 위해서는 대용량의 하드디스크를 필요로 할 뿐만 아니라, 이와 같이 특정 형식(예로 VCD 형식)으로 하드디스크에 저장된 데이터를 컴퓨터로 읽어봐서 파형분석기(waveform viewer)로 분석하는 과정에서도 매우 긴 시간이 필요하게 되어 이 또한 전체 검증 시간을 증가시키게 된다. 본 특허에서 제안되는 방법은 1차 시뮬레이션 수행을 설계 코드에 존재하는 모든 신호선들과 변수들을 덤프하지는 않도록 함으로서 시뮬레이션 시간의 증가를 초래하지 않고 신속하게 시뮬레이

션을 진행할 수 있게 한다. 이와 같은 1차 시뮬레이션 과정에서 수행되어지는 것은 1차 이후의 추가 시뮬레이션들을 기존방법에서와 같은 시뮬레이션 시간 0에서 수행되지 않고 사용자가 관심이 있는 시뮬레이션 시간대에서 아주 가까운 곳에서부터 시작할 수 있도록 시뮬레이션 일정 간격(예로 시뮬레이션 시작에서부터 매 10,000 나노초마다, 혹은 매 5,000 시뮬레이션 사이클마다)마다 혹은 원하는 시점들 t_0, t_1, ... t_t마다 시뮬레이션의 상태 s_0, s_1, ... s_t를 저장시키는 것이다. 이와 같이 1 이상의 시점들에서 시뮬레이션의 상태를 저장하게 되면, 이후부터는 재 시뮬레이션을 시뮬레이션 시간 0에서뿐만 아니라 이 각 시점들에서부터도 가능하게 된다. 따라서 1차 시뮬레이션을 통하여 설계코드에 존재하는 시그널들이나 변수들 중에서 설계 오류의 위치를 찾기 위하여 탐침이 필요한 특정 시그널들이나 변수들을 선정(필요한 경우에는 설계 코드에 존재하는 모든 시그널들과 변수들을 선정)하고 이들 탐침 대상에 대한 탐침이 어느 시뮬레이션 시간대에서 필요한 가를 파악한 다음, 2차 이후의 시뮬레이션들에서는 1차 시뮬레이션에서 저장된 1 이상의 시뮬레이션 상태들에서 이 저장이 일어난 시뮬레이션 시간이 탐침이 필요한 시뮬레이션 시간대(t_s , t_e)의 종료시점 t_e에서 시뮬레이션 시간적으로 앞서면서(시뮬레이션 시간 10,000 나노초와 20,000 나노초가 있는 경우에, 10,000 나노초는 20,000 나노초에 비하여 시간적으로 앞서 있다고 하고, 20,000 나노초는 10,000 나노초에 비하여 시간적으로 뒤에 있다고 함) 제일 가까운 곳에 있는 시뮬레이션 상태 S_i로 시뮬레이션

의 상태를 설정하고 시뮬레이션 시간 t_i에서부터 t_e까지 시뮬레이션을 수행하면서 해당 탐침 대상에 대한 탐침을 진행하고, 필요시에는 S_i보다 시간적으로 앞선 시뮬레이션 상태들 S_{i-1}, S_{i-2}, ... S_{i-n} 각각으로 시뮬레이션의 상태를 순차적을 설정하면서 각각의 시뮬레이션 시간 t_{i-1}, t_{i-2} , ... t_{i-n}에서부터 t_i, t_{i-1}, ... t_{i-n+1}까지 순서적으로 추가적인 n번의 시뮬레이션을 순차적으로 설계 코드에서 설계 오류의 위치와 원인이 밝혀 질 때까지 진행한다. 따라서 이와같은 과정을 통하여 설계에 대한 디버깅을 신속하게 할 수 있는데, 이와 같은 과정은 기존의 시뮬레이션 기반의 검증 방법들에 비하여 검증 시간을 크게 단축시켜 줄수 있음으로 매우 효율적인 검증 방법이 된다. 이와 같은 시뮬레이션 방법을 A 시뮬레이션 방법이라고 칭한다.

<21> 또 다른 방법으로서는 설계 코드에서 DUV(Design Under Verification)이나 테스트벤치에 존재하는 계층 구조를 이용한 분할 및 정복(divide & conquer) 방법을 사용할 수 있다. 이 경우에는 검증 소프트웨어가 설계 코드를 읽어 들여서 DUV와 테스트벤치에 대하여 분할을 수행하여 설계 코드를 2 이상의 설계블록들로(M개의 설계블록들로 분할되었다고 가정) 나누고, 이 설계블록 각각에 존재하는 모든 입력과 입출력들을 탐침 대상으로 선정한 후에 이들 탐침 대상들이 1차 시뮬레이션 과정에서 덤프되어질 수 있도록 설계 코드에 코드를 부가한다. 여기서 설계블럭(design block)들이란 크게 DUV 뿐만 아니라 테스트벤치 모두를 포함한다. 뿐만 아니라, 이들 DUV와 테스트벤치는 일반적으로 계층적인 구조로 내부에 다양한 1 이상의 하위모듈들을 가지고 있는데, 이들 하위모듈 각각도 설계블럭이라고 할 수

있다. 1차 시뮬레이션을 수행하면서 각 설계블록들의 모든 입력과 입출력들을 덤프하여 파일 형태로 저장한 후에, 이를 검증 소프트웨어를 이용하여 해당 설계블록들에 대한 M개의 테스트벤치들로 변환하여 해당 설계블록의 설계 코드와 같이 해당 테스트벤치를 같이 시뮬레이션 컴파일을 수행하여 M개의 설계블록들에 대응되는 M개의 시뮬레이션 실행파일을 생성한다. 이와 같이 각 설계블록별로 새로운 테스트벤치를 생성하여 시뮬레이션 컴파일을 수행하여 별도의 시뮬레이션 실행파일을 생성하게 되는 것에 비하여, 해당 설계블록들의 모든 입력들과 입출력들을 탐침된 파일(예로 VCD 파일)을 테스트벤치화 하지 않고 직접 이용하게 되면 변환 시간과 시뮬레이션 컴파일 시간을 절약할 수 있지만 해당 설계블록을 추후에 시뮬레이션 하는 경우에 필요한 입력값들을 상기 탐침된 파일에서 읽어서 인가해주기 위하여서는 API(Application Program Interface)를 사용하는 것(Verilog에서는 PLI, VHDL에서는 FLI 등)이 필요하다. 그러나 이와 같은 API를 사용하여 별도의 코드를 시뮬레이터에 연동시키게 되면 API 오버헤드로 인하여 시뮬레이션의 속도가 크게 저하됨으로 득보다 실이 많다. 그러나 최근에 사용되는 특정 시뮬레이터(예로 Synopsys의 VCS 7.0이상)에서는 API를 사용하지 않고 외부 코드모듈(예로 C, C++ 코드)를 시뮬레이터에 연동시키는 것이 가능하기도 한데, 이와 같은 경우에는 직접 1차 시뮬레이션에서 덤프를 통하여 얻어진 탐침 파일을 직접 이용하여 시뮬레이션 컴파일을 수행하여 M개의 시뮬레이션 실행 파일을 생성하는 것도 가능하다. 이 후에 설계 오류에 대한 위치를 파악하기 위하여 추가적인 시그널들이나 변수들에 대한 탐침이 필요한 경우에는 상기 M개의 시뮬레이션 실행파일들 중에서 해당 시그널들이나 변

수들이 존재하는 해당 설계블록과 상기 1차 시뮬레이션 후에 생성된 덤프 파일이나 테스트벤치로써 시뮬레이션 컴파일하여 얻어진 특정 시뮬레이션 실행파일을 이용한 추가적인 시물레이션을 하게 된다. 이와 같은 추가적인 시뮬레이션에서는 해당 설계블록들에 존재하는 모든 시그널들이나 변수들에 대한 탐침을 수행하면서 시뮬레이션을 진행하거나, 또는 관심대상이 되는 특정 시그널들이나 변수들에 대한 탐침을 수행하면서 시뮬레이션을 진행한다. 만일, 탐침이 필요한 특정 시그널들이나 변수들이 2 이상의 설계블록들에 분산되어져서 존재하는 경우에는 이들 설계블록들에 대하여 상기 설계블록이 하나인 과정과 동일한 과정들을 순서적으로 반복하여 수행하여 각 과정에서 얻어진 탐침 결과들을 통합하면 된다. 이와 같이 DUV와 테스트벤치 전체를 시뮬레이션하지 않고서 특정 설계블록과 덤프 파일이나 새롭게 생성된 테스트벤치만을 이용한 시뮬레이션을 수행하게 되면 시뮬레이션 속도를 크게 증가시킬 수 있다. 이와 같은 시뮬레이션 방법은 B 시뮬레이션 방법이라 칭한다.

<22> 이와 같은 두가지 방법들은 기존의 시뮬레이션 방법들에 비하여 별도의 특별한 하드웨어 기반의 검증 플랫폼들(예로 하드웨어 에뮬레이터 또는 FPGA 프로토타이핑 플랫폼 등)을 사용하지 않고서도 모두 시뮬레이션의 속도를 크게 향상시킬 수 있다. 물론 상기 두가지 방법들을 병행한 검증 방법도 가능하다.

<23> 그러나 위에서 언급된 A 시뮬레이션 방법과 B 시뮬레이션 방법의 경우에 다음과 같은 문제점들이 있을 수 있다. 우선 A 시뮬레이션 방법의 경우에 2차 이후의 시뮬레이션에서 시뮬레이션 하고자 하는 시간대 (t_s, t_e)가 매우 긴 경우에 t_i, t_{i-1},

t_{i-2}, ... t_{i-n} 시점에서의 총 n+1 횟수의 시뮬레이션을 순차적으로 수행하여야 함으로서 원래 시뮬레이션 방식보다는 검증 시간을 단축할 수 있지만 그래도 많은 검증 시간이 A 시뮬레이션 방법에 필요하게 된다. 또한 B 시뮬레이션 방법의 경우에도 2차 이후의 시뮬레이션에서 시뮬레이션 하여야 하는 설계블록들의 숫자가 큰 경우에는 이들을 순차적으로 수행하여야 함으로서 원래 시뮬레이션 방식보다는 검증 시간을 단축할 수 있지만 그래도 많은 검증 시간이 B 시뮬레이션 방법에도 필요하게 된다. 그러나, 시뮬레이터가 2 이상이고 이들 시뮬레이터들이 수행되는 복수개의 컴퓨터들(예로 X개의 시뮬레이터가 X개의 컴퓨터에 인스톨되어져 있음)이 네트워크로 연결되어져 있는 경우에는 이와 같은 A 시뮬레이션 방법이나 B 시뮬레이션 방법에서 수행되는 1차 이후의 추가 시뮬레이션들을 동시에 병렬적으로 수행하는 것이 가능하다. 이와 같은 병렬적 시뮬레이션은 이 병렬적으로 수행되어져야 하는 각 시뮬레이션들이 완전히 독립적으로 수행하는 것이 가능함으로서 1차 이후의 추가 시뮬레이션의 수행을 획기적으로 빠르게 수행하는 것이 가능하다.

<24> 1차 시뮬레이션은 가능한 최대한도로 빠르게 수행하면서도, 1차 이후의 1회 이상의 시뮬레이션을 수행하는데 필요한 정보를 수집하는 것이 매우 중요하다. 이를 위하여서는 여러 가지 방법들을 생각할 수 있다. 그 중에서 첫째 방법은 1차 시뮬레이션도 2 이상의 시뮬레이터들을 사용하여서 병렬적으로 수행하는 것이다. 이와 같은 1차 시뮬레이션에서의 병렬 수행은 각 시뮬레이터가 독립적으로 수행되는 것이 아니고 서로 연동되면서 수행하여야 한다. 둘째 방법은 1차 시뮬레이션에서는

이벤트-기반(event-driven) 시뮬레이터를 사용하는 대신에 사이클-기반(cycle-based) 시뮬레이터를 사용하는 것이다. 셋째 방법은 Verilog나 VHDL로 코딩된 설계 코드를 자동화된 방법(예로 HDL2SystemC 변환툴을 사용하여서 자동 변환)이나 수동으로 SystemC 코드로 변환시켜서 HDL 시뮬레이터 대신에 SystemC 시뮬레이터를 사용하는 것이다. 일반적으로 SystemC 시뮬레이터는 Verilog 시뮬레이터나 VHDL 시뮬레이터와 비교하여 매우 빠르게 수행하는 것이 가능하다. 둘째와 셋째 방법의 경우에는 1차 시뮬레이션을 SystemC 시뮬레이터나 사이클-기반 시뮬레이터로써 수행하면서 시뮬레이션의 상태를 1회 이상 저장하는 대신에 설계 코드의 상태정보(상태정보란 임의의 검증 실행 시점에서 설계 코드에 존재하는 모든 메모리소자들의 값)를 1회 이상의 특정 시점에서 1회 이상 저장하고, 1차 이후의 1회 이상의 시뮬레이션을 2 이상의 HDL 시뮬레이터를 사용하고 병렬적으로 수행하는데 상기 1차 이후의 1회 이상의 시뮬레이션 각각을 상기 1차 시뮬레이션 수행 과정에서 저장된 1 이상의 설계 코드 상태정보로 초기상태를 다르게 설정하여서 수행한다. 1차 이후에 수행되는 시뮬레이터에 사용되는 테스트벤치는 1차 시뮬레이션에서 사용한 테스트벤치(예로 SystemC로 구술된 테스트벤치)를 그대로 사용할 수 있을 뿐만 아니라, 둘째와 셋째방법에서는 1차 시뮬레이션에서 사용되는 시뮬레이터와 1차 이후 시뮬레이션에서 사용되는 시뮬레이터가 서로 상이함으로 시뮬레이션 상태를 이용하는 것이 불가능함으로 설계 코드의 상태정보를 이용하여야 함과 더불어 DUV의 입력값과 입출력값들도 1차 시뮬레이션 전 과정에 걸쳐서 탐침하고 테스트벤치화 해서 1차 이후의 시뮬레이션에서 사용할 수도 있다.

<25> 뿐만 아니라, 위에서 언급한 기법은 RTL 설계 코드를 이용하여 RTL에서 시뮬레이션한 결과를 RTL 설계 코드를 합성(synthesis)하여서 게이트-네트리스트를 생성하고, 필요시에는 배치 및 배선 결과에 의한 지연시간 정보까지를 가져와서 수행되는 게이트수준에서의 시뮬레이션 결과에 이용함으로서 게이트수준에서의 시뮬레이션 시에서의 디버깅을 매우 효과적으로 하는 것에 이용할 수 있다. 즉, 1차 시뮬레이션을 RTL 설계 코드를 이용하여 RTL에서 시뮬레이션하면서 설계 코드에 대한 상태정보를 1 이상의 시뮬레이션 시점에서 저장하고, 1차 이후의 1회 이상의 시뮬레이션은 게이트수준에서의 시뮬레이션을 상기 RTL 시뮬레이션에서 저장된 1 이상의 상태정보들 각각을 이용함으로서 게이트수준에서의 시뮬레이션을 병렬적으로 수행하는 것도 가능하다.

<26> 상기 목적 외에 본 발명의 다른 목적 및 이점들은 첨부한 도면을 참조한 실시 예에 대한 상세한 설명을 통하여 명백하게 드러나게 될 것이다.

<27> 도1 은, 컴퓨터에서 운영되는 본 발명의 검증 소프트웨어와 시뮬레이터를 갖는 컴퓨터로 구성된 본 발명에 관한 설계 검증 장치의 일 예를 개략적으로 도시한 도면이다.

<28> 도2 는, 컴퓨터에서 운영되는 본 발명의 검증 소프트웨어와 시뮬레이터를 갖는 2 이상의 컴퓨터들과 이들 컴퓨터들을 연결하는 컴퓨터 네트워크로 구성된 본 발명에 관한 설계 검증 장치의 또 다른 일 예를 개략적으로 도시한 도면이다.

<29> 도3 은 1차 시뮬레이션 수행 도중에 주기적으로 시뮬레이션의 상태를 저장하고, 2차 시뮬레이션 수행에서 1차 시뮬레이션에서 저장된 총 n+1개의 시뮬레이션

상태 중에서 특정 상태 s_{i-1}로 시뮬레이션을 설정하여 t_{i-1}에서부터 t_i까지 시뮬레이션을 수행하는 과정을 개략적으로 도시한 것이다.

<30> 도4 는 1차 시뮬레이션 수행 도중에 주기적으로 시뮬레이션의 상태를 저장하고, 1차 이후의 시뮬레이션에서 상기 1차 시뮬레이션에서 저장된 1 이상의 시뮬레이션 상태들에서 이 저장이 일어난 시물레이션 시간이 탐침이 필요한 시뮬레이션 시간대 (t_s, t_e)의 종료시점 t_e에서 시뮬레이션 시간적으로 앞서면서 제일 가까운 곳에 있는 시뮬레이션 상태 S_i로 시뮬레이션의 상태를 설정하고 시뮬레이션 시간 t_i에서부터 t_e까지 시뮬레이션을 수행하면서 해당 탐침 대상에 대한 탐침을 진행하고, 필요시에는 S_i보다 시간적으로 앞선 시뮬레이션 상태들 S_{i-1}, S_{i-2}, ... S_{i-n} 각각으로 시뮬레이션의 상태를 순차적을 설정하면서 각각의 시뮬레이션 시간 t_{i-1}, t_{i-2}, ... t_{i-n}에서부터 t_i, t_{i-1}, ... t_{i-n+1}까지 순서적으로 추가적인 n번의 시뮬레이션을 순차적으로 진행하는 과정을 개략적으로 도시한 것이다.

<31> 도5(a) 은 도1 과 같은 장치를 이용한 설계 검증을 수행하는 과정을 개략적으로 도시한 순서도이다.

<32> 도5(b) 는 도2 와 같은 장치를 이용한 설계 검증을 수행하는 과정을 개략적으로 도시한 순서도이다.

<33> 도6(a) 는 도2 와 같은 장치를 이용한 설계 검증에서 1차 이후의 시뮬레이션 과정에서 1차 시뮬레이션 과정에서 저장된 2 이상의 시뮬레이션 상태들 중에서 2

이상의 시뮬레이션 상태를 선택하여 이들 2 이상의 시뮬레이션 상태로 2 이상의 시뮬레이터들에 시뮬레이션을 설정하여 이 2 이상의 시뮬레이션을 병렬적으로 수행하고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공하는 과정을 도시한 그림이다.

<34> 도6(b) 는 도2 와 같은 장치를 이용한 설계 검증에서 1차 이후의 시뮬레이션 과정에서 1차 시뮬레이션 과정에서 저장된 2 이상의 설계 코드의 상태정보들 중에서 2 이상의 상태정보들을 선택하여 이들 2 이상의 설계 코드의 상태정보들 각각으로 2 이상의 시뮬레이터들에서 병렬적으로 수행되는 설계 코드들 각각의 초기 상태를 설정하여 이 2 이상의 시뮬레이션을 병렬적으로 수행하고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공하는 과정을 도시한 것인데, 1차 시뮬레이션을 RTL 시뮬레이션으로 수행하고 1차 이후의 시뮬레이션을 게이트수준 시뮬레이션으로 수행하는 과정을 도시한 그림이다.

<35> 도2 와 같은 장치를 이용한 설계 검증에서 1차 이후의 시뮬레이션과정에서 1차 시뮬레이션 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 시뮬레이션 전과정에서 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하여 이를 2 이상의 컴퓨터들에서 병렬적으로 실행시키고 이들 시뮬레이션 결과들을 취합하여 사용자에게 제공할 수 있다. 그러나 이와 같은 방법은 1차 시뮬레이션 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값

들을 시뮬레이션 전과정에서 탐침하게 됨으로 시뮬레이션의 속도가 크게 떨어질 수 있다. 이를 개선하는 방법을 도6(c)로써 설명한다.

<36> 도6(c) 는 도2 와 같은 장치를 이용한 설계 검증에서 1차 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점들에서 2 이상의 시뮬레이션 상태를 저장하고, 1차 이후에 처음으로 수행되는 시뮬레이션에서 2 이상의 시뮬레이터를 이용하여 상기 1차 시뮬레이션 과정에서 저장된 2 이상의 시뮬레이션 상태로부터 시물레이션이 병렬적으로 진행할 수 있도록 하고, 이 병렬적 시뮬레이션이 진행되는 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 병렬적으로 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하고, 상기 2 이상의 컴퓨터들에서 병렬적으로 시뮬레이션 하는 과정을 도시한 그림이다.

【발명의 효과】

<37> 상술한 바와 같이, 본 발명에 따른 검증 장치 및 이를 이용한 설계 검증 방법의 목적은 초대규모급 설계 검증을 위하여 시뮬레이션을 수행하는 경우에 1차 시뮬레이션을 통해서 1차 이후에 추가적으로 진행될 시뮬레이션들을 매우 빠르고 효과적으로 진행하는 것에 필요한 최소한의 정보들을 수집하고, 이 후에 진행되는 추가적인 시뮬레이션들을 하나의 시뮬레이터를 이용하여 순차적으로, 내지는 2 이상의 시뮬레이터를 이용하여 병렬적으로 빠르게 진행하게 함으로서 전체의 시뮬레이션 시간을 단축하며, 빠른 시간 내에 설계 코드에 존재하는 오류들의 위치를 정확

히 찾아내어 수정하는 것이 가능하다.

<38> 이상 설명한 내용을 통해 당업자라면 본 발명의 기술사상을 일탈하지 아니하는 범위에서 다양한 변경 및 수정이 가능함을 알 수 있을 것이다. 따라서, 본 발명의 기술적 범위는 실시 예에 기재된 내용으로 한정되는 것이 아니라 특허 청구의 범위에 의하여 정하여져야만 한다.

【특허청구범위】

【청구항 1】

검증 소프트웨어와 1 이상의 시뮬레이터를 구비하는 설계검증 장치에 있어서,

상기 검증 소프트웨어는 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 자동화된 방식을 통하여 부가 코드나 부가 회로를 추가하여 1차 시뮬레이션을 수행하면서 1차 시뮬레이션 이후에 수행되는 1 회 이상의 시뮬레이션들을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 하고, 상기 1 이상의 시뮬레이터를 이용한 1차 시뮬레이션을 수행하면서 상기 최소한의 정보를 수집하고, 이 수집된 정보를 이용하여 1차 이후의 1회 이상의 시뮬레이션들을 신속하게 수행하는 것을 가능하게 하는 설계 검증 장치.

【청구항 2】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를 알아내고 이를 수정하는 설계 검증 방법에 있어서,

상기 수 차례의 시뮬레이션들에서의 각각의 시뮬레이션 과정 하나 하나를 1차 시뮬레이션과 1차 이후의 1회 이상의 시뮬레이션들로 나누어서 수행하며, 상기

1차 시뮬레이션을 수행하면서 상기 1차 이후의 1회 이상의 시뮬레이션들을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 부가 코드를 검증 소프트웨어를 이용하여 자동화된 방식으로 추가하고, 상기 1차 시뮬레이션을 수행하면서 상기 최소한의 정보를 수집하고, 이 수집된 정보를 이용하여 상기 1차 이후의 1회 이상의 시뮬레이션들을 신속하게 수행하는 것을 가능하게 하는 검증 방법.

【청구항 3】

제 2 항 내지는 제 12 항에 있어서,

최소한의 정보 수집이 1차 시뮬레이션 과정에서 일정 간격으로 혹은 특정 시점들에서 시뮬레이션의 상태를 1 이상의 파일 형태로 저장하는 설계 검증 방법

【청구항 4】

제 2 항 내지는 제 12 항에 있어서,

최소한의 정보 수집이 1차 시뮬레이션 과정에서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들의 입력과 입출력을 시뮬레이션 전체 과정에 걸쳐서 지속적으로 탐침하여 1 이상의 파일 형태로 저장하는 설계 검증 방법.

【청구항 5】

제 3 항에 있어서,

1차 시뮬레이션 시행 중에 1 이상의 시점에서 저장된 1 이상의 시뮬레이션 상태들 중에서 1 이상의 시뮬레이션 상태를 선정하여 1차 이후의 1회 이상의 시뮬레이션들을 상기 선정된 1 이상의 시뮬레이션 상태에서부터 시작될 수 있도록 시뮬레이터를 1회 이상 설정한 후에 시뮬레이션을 1회 이상 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행함으로서 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 6】

제 5 항에 있어서,

1차 이후의 시뮬레이션들을 상기 선정된 2 이상의 시뮬레이션 상태들에서부터 시작될 수 있도록 하나의 시뮬레이터를 2회 이상 순서적으로 설정하고 2회 이상 순서적으로 시뮬레이션 진행하는 과정에 있어서, 1차 시뮬레이션 수행 시에 시뮬레이션의 상태를 저장하는 시점이 시뮬레이션 시간적으로 제일 뒤에 있는 시뮬레이션 상태를 우선 상기 하나의 시뮬레이터에 설정하여 이 시뮬레이션의 상태를 가지고 시뮬레이션을 진행하고, 이 후의 시뮬레이션들도 상기 단일 시뮬레이터로써 수행하는 과정에서 상태를 저장한 시점이 시뮬레이션 시간적으로 뒤에 있는 것들을 우선으로 하여 시뮬레이터에 설정하여 시뮬레이션들이 시뮬레이션 시간적으로 제일 뒤

에서부터 앞서는 순서대로 진행하면서 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들에 대한 탐침을 수행함으로서 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 7】

제 2 항 내지는 제 12 항 내지는 제 3 항 내지는 제 5 항 내지는 제 6 항에 있어서,

1차 시뮬레이션 시행 중에 1 이상의 시점에서 시뮬레이션 상태를 저장하는 방법을 시뮬레이터의 save 명령어 혹은 checkpoint 명령어를 사용하고, 1차 이후의 1회 이상의 시뮬레이션들을 상기 선정된 1 이상의 시뮬레이션 상태에서부터 시작될 수 있도록 시뮬레이터를 1회 이상 설정한 후에 시뮬레이션을 진행하는 방법을 시뮬레이터의 restart 명령어 혹은 restore 명령어를 사용함으로서 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 8】

제 4 항에 있어서,

1차 시뮬레이션 실행 중에 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 대한 입력과 입출력들을 시뮬레이션 전 과정에서 탐침하여 저장한 1 이상의 탐침 파일들 중에서 1 이상을 선정하여 1차 이후의 시뮬레이션들을 상기 선정된 1 이상의 탐침 파일과 해당 1 이상의 설계블록들

을 이용하여 시뮬레이션 컴파일하여 생성된 1 이상의 시뮬레이션 실행 파일들 중에서 추가적인 탐침이 필요한 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 시그널들과 변수들을 가지고 있는 해당 1 이상의 설계블록들을 컴파일된 형태로 가지고 있는 1 이상의 시뮬레이션 실행 파일을 실행시키면서 상기 추가적인 탐침을 수행함으로서 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 9】

제 2 항 내지는 제 12 항 내지는 제 3 항 내지는 5 항 내지는 제 6 항 내지는 제 7 항에 있어서,

1차 이후의 1회 이상의 시뮬레이션을 1차 시뮬레이션 과정 중에서 저장된 2 이상의 시뮬레이션 상태들 각각을 이용하여 2회 이상의 부분 시뮬레이션을 하는 것으로 전환하여, 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 상기 2 이상의 시뮬레이터 상태들 각각을 상기 2 이상의 시뮬레이터 각각에 설정하여 2 이상의 시뮬레이션들이 서로 독립적으로 상기 2 이상의 시뮬레이터로써 병렬적으로 실행하여 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 10】

제 2 항 내지는 제 12 항 내지는 제 4 항 내지는 8 항에 있어서,

1차 이후의 1회 이상의 시뮬레이션을 1차 시뮬레이션을 통하여 얻어진 2 이

상의 설계블록들과 이들 설계블록들 각각의 입력과 입출력을 1차 시뮬레이션 전과정에서 탐침한 2 이상의 탐침 파일들을 시뮬레이션 컴파일하여 얻어진 2 이상의 시뮬레이션 실행 파일들을 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 독립적으로 병렬적으로 실행하여 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 11】

제 2 항 내지는 제 12 항 내지는 제 4 항 내지는 8 항에 있어서,

1차 이후의 1회 이상의 시뮬레이션을 1차 시뮬레이션을 통하여 얻어진 2 이상의 설계블록들과 이들 설계블록들 각각의 입력과 입출력을 1차 시뮬레이션 전과정에서 탐침한 2 이상의 탐침 파일들을 변환한 2 이상의 테스트벤치 파일들을 시뮬레이션 컴파일하여 얻어진 2 이상의 시뮬레이션 실행 파일들을 컴퓨터 네트워크로 연결된 2 이상의 컴퓨터에 인스톨된 2 이상의 시뮬레이터들을 이용하여 독립적으로 병렬적으로 실행하여 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【청구항 12】

임의의 시뮬레이션 수행 내지는 시뮬레이션가속 수행 내지는 하드웨어에뮬레이션 수행 내지는 프로토타이핑 수행을 통하여 얻어진 결과를 이용한 추가적인 시뮬레이션 수행에서,

상기 추가적인 시뮬레이션 수행을 1차 1회의 시뮬레이션과 1차 이후의 1회 이상의 시뮬레이션들로 나누어서 수행하며, 상기 1차 시뮬레이션을 수행하면서 상기 1차 이후의 1회 이상의 시뮬레이션을 시뮬레이션 구간들이나 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계블록들에 한정하여 시뮬레이션들이 이루어질 수 있도록 하는데 필요한 최소한의 정보를 1차 시뮬레이션 과정에서 자동적으로 수집할 수 있도록 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 부가 코드를 검증 소프트웨어를 이용하여 자동화된 방식으로 추가하고, 상기 1차 시뮬레이션을 수행하면서 상기 최소한의 정보를 수집하고, 이 수집된 정보를 이용하여 상기 1차 이후의 1회 이상의 시뮬레이션을 신속하게 수행하는 것을 가능하게 하는 검증 방법.

【청구항 13】

제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항에 있어서,

1차의 시뮬레이션을 병렬 시뮬레이션으로 수행하는 검증 방법

【청구항 14】

제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항에 있어서,

1차의 시뮬레이션을 사이클-기반 시뮬레이션으로 수행하고, 1차 이후의 1회 이상의 시뮬레이션은 이벤트-기반 시뮬레이션으로 수행하는 검증 방법

【청구항 15】

제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항에 있어서,

1차의 시뮬레이션을 SystemC 시뮬레이터를 이용하여 수행하고, 1차 이후의 1회 이상의 시뮬레이션은 Verilog 시뮬레이터 내지는 VHDL 시뮬레이터를 이용하여 수행하는 검증 방법

【청구항 16】

제 2 항 내지는 제 3 항 내지는 제 4 항 내지는 제 5 항 내지는 제 9 항 내지는 제 10 항 내지는 제 11 항 내지는 제 12 항에 있어서,

1차의 시뮬레이션을 RTL 시뮬레이션으로 수행하고, 1차 이후의 1회 이상의 시뮬레이션은 게이트수준 시뮬레이션으로 수행하는 검증 방법

【청구항 17】

제 14 항 내지는 제 15 항 내지는 제 16 항에 있어서,

상기 1차 이후의 1회 이상의 시뮬레이션을 2 이상의 시뮬레이터들을 이용하여서 병렬적으로 진행하는 검증 방법

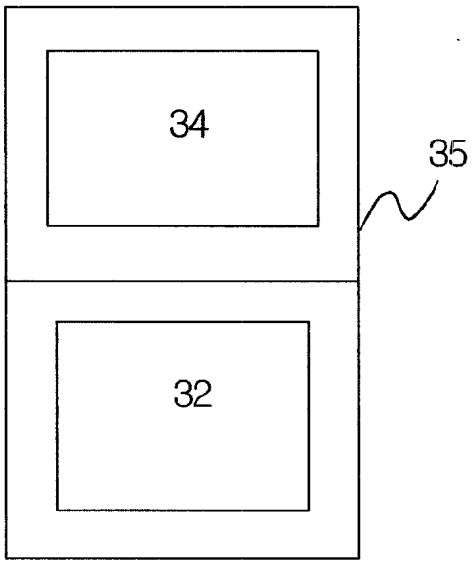
【청구항 18】

시뮬레이션을 여러 개의 테스트벤치로써 수 차례 수행하여 설계 코드나 합성으로 생성된 게이트수준의 네트리스트에 존재하는 1 이상의 설계 오류들의 위치를 알아내고 이를 수정하는 설계 검증 방법에 있어서,

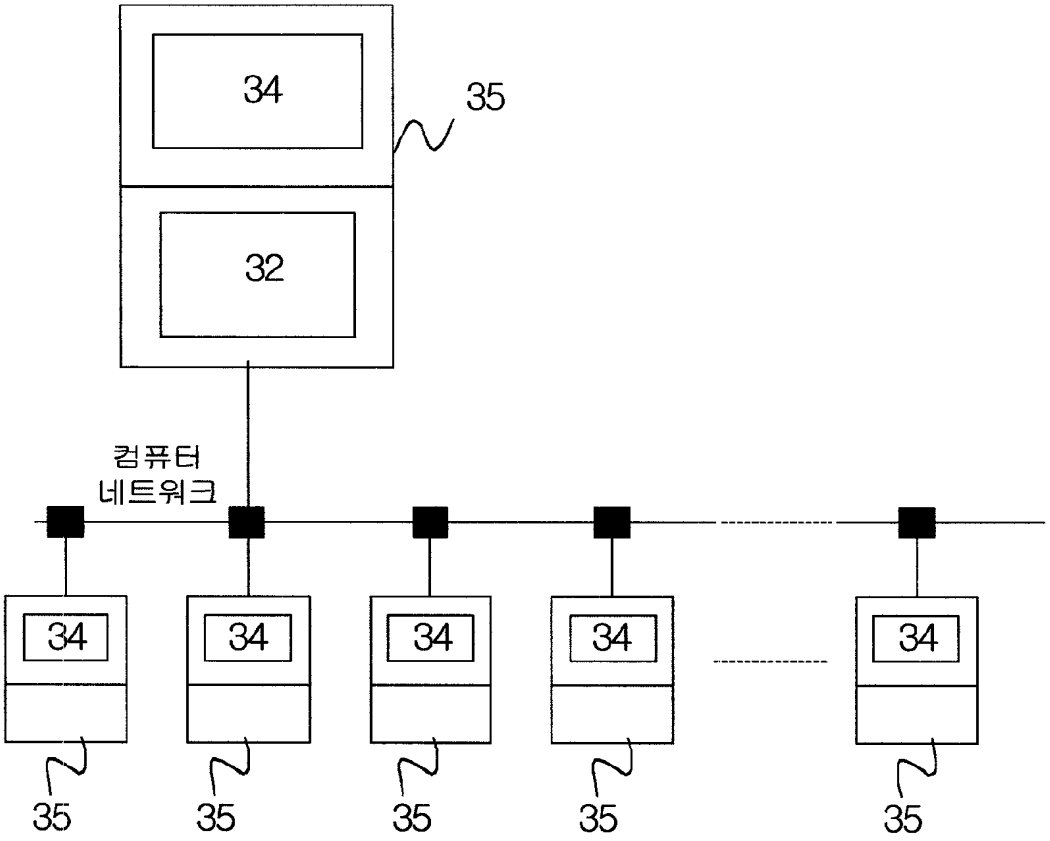
설계 검증의 1차 시뮬레이션 과정에서 2 이상의 시뮬레이션 시점들에서 2 이상의 시뮬레이션 상태를 저장하고, 1차 이후에 처음으로 수행되는 시뮬레이션에서 2 이상의 시뮬레이터를 이용하여 상기 1차 시뮬레이션 과정에서 저장된 2 이상의 시뮬레이션 상태로부터 시뮬레이션이 병렬적으로 진행할 수 있도록 하고, 이 병렬적 시뮬레이션이 진행되는 과정에서 설계 코드에 존재하는 1 이상의 특정 설계블록들의 입력과 입출력값들을 병렬적으로 탐침하여 얻어진 2 이상의 덤프 파일들이나 이들을 테스트벤치로 변환된 파일들을 해당 설계블록들과 같이 2 이상의 시뮬레이터들을 이용하여 동시에 시뮬레이션 컴파일하여 2 이상의 시뮬레이션 실행파일들을 생성하고, 상기 2 이상의 컴퓨터들에서 병렬적으로 시뮬레이션 하는 과정을 통하여 시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법시뮬레이션 속도를 크게 증가시키지 않으면서도 설계블럭들에 대한 가시도를 확보하는 설계 검증 방법

【도면】

【도 1】

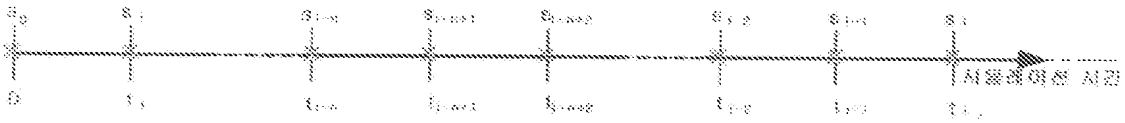


【도 2】



【도 3】

i) 1차 시뮬레이션

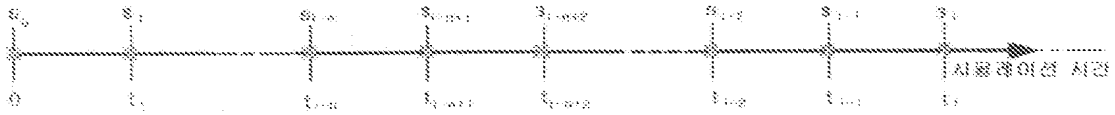


ii) 1차 이후의 2차 시뮬레이션

【도 4】

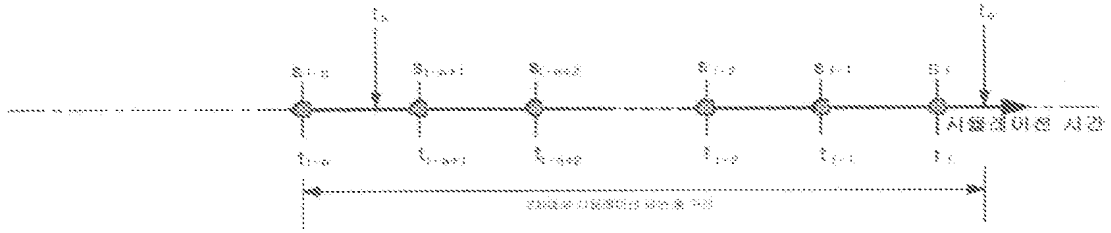
i) 1차 시뮬레이션

※ 시뮬레이션 실제 적용 사항

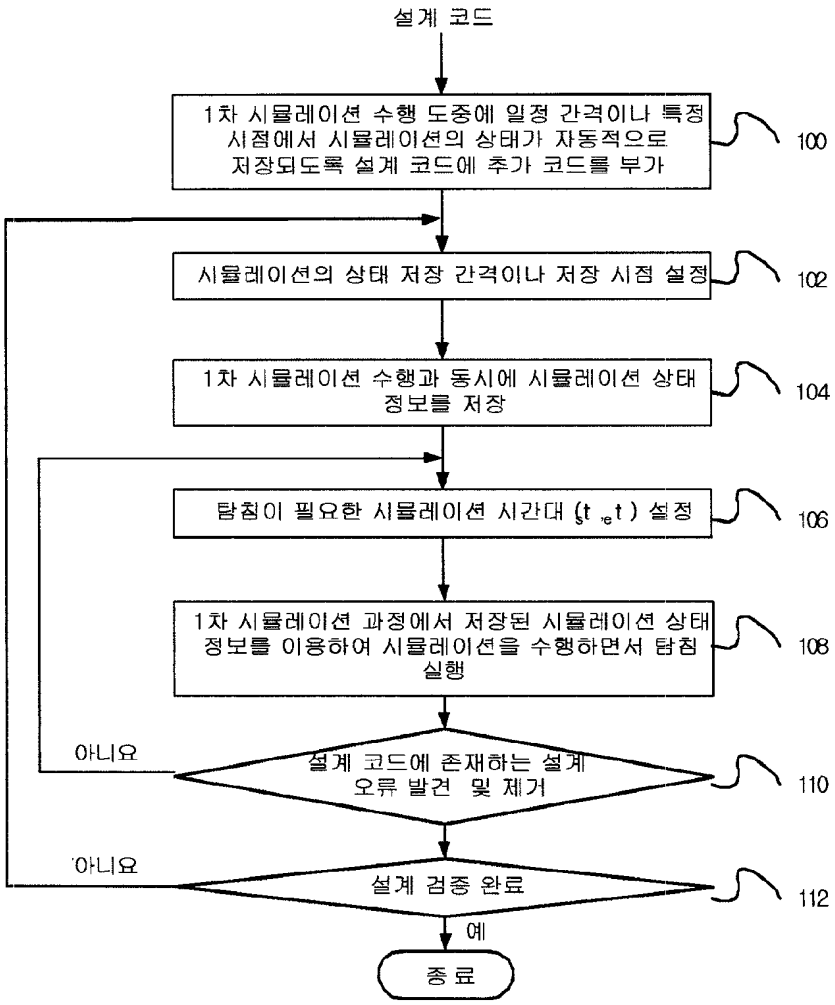


ii) 1차 이후의 2차 시뮬레이션

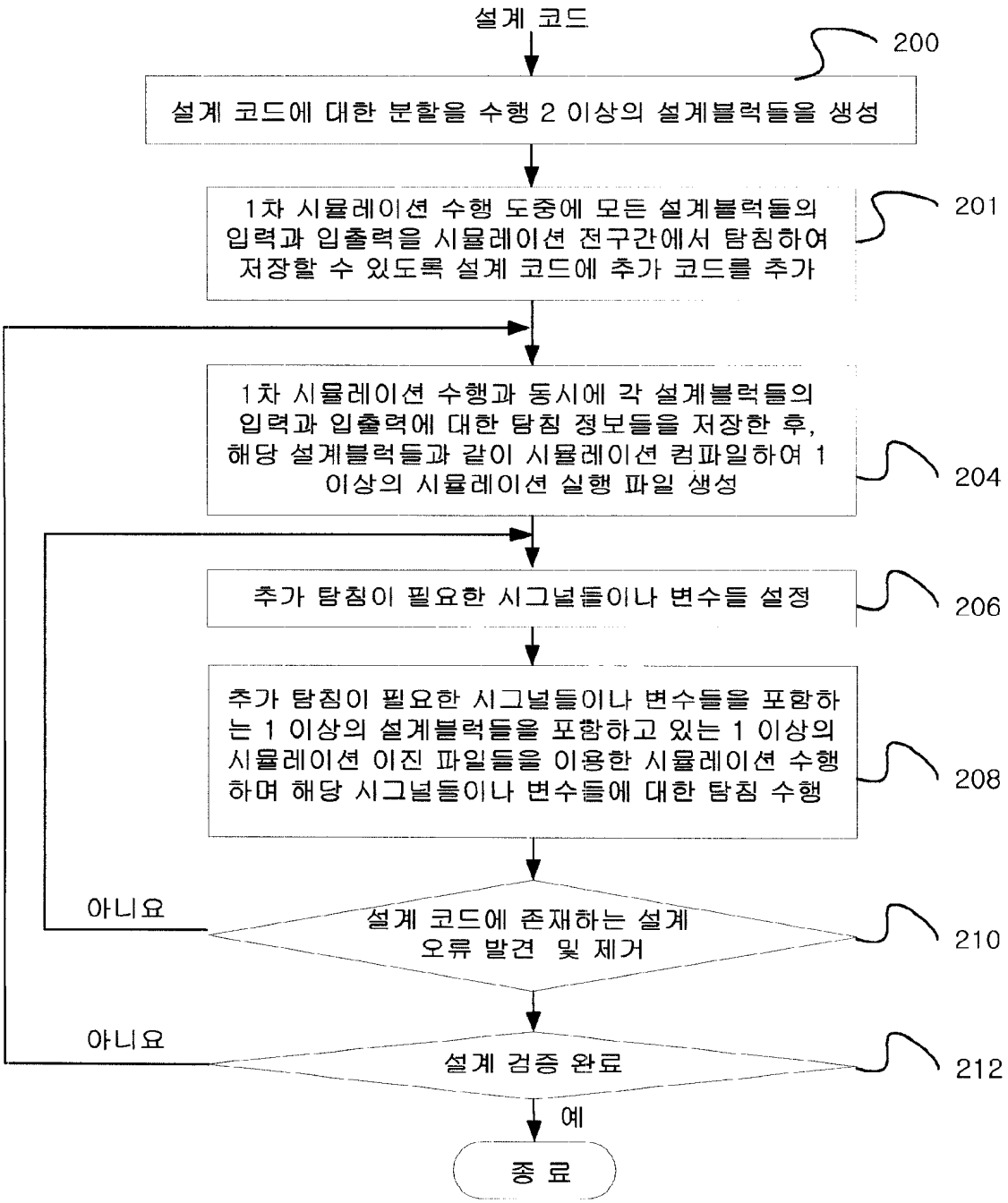
● 시뮬레이션 실제 적용 사항



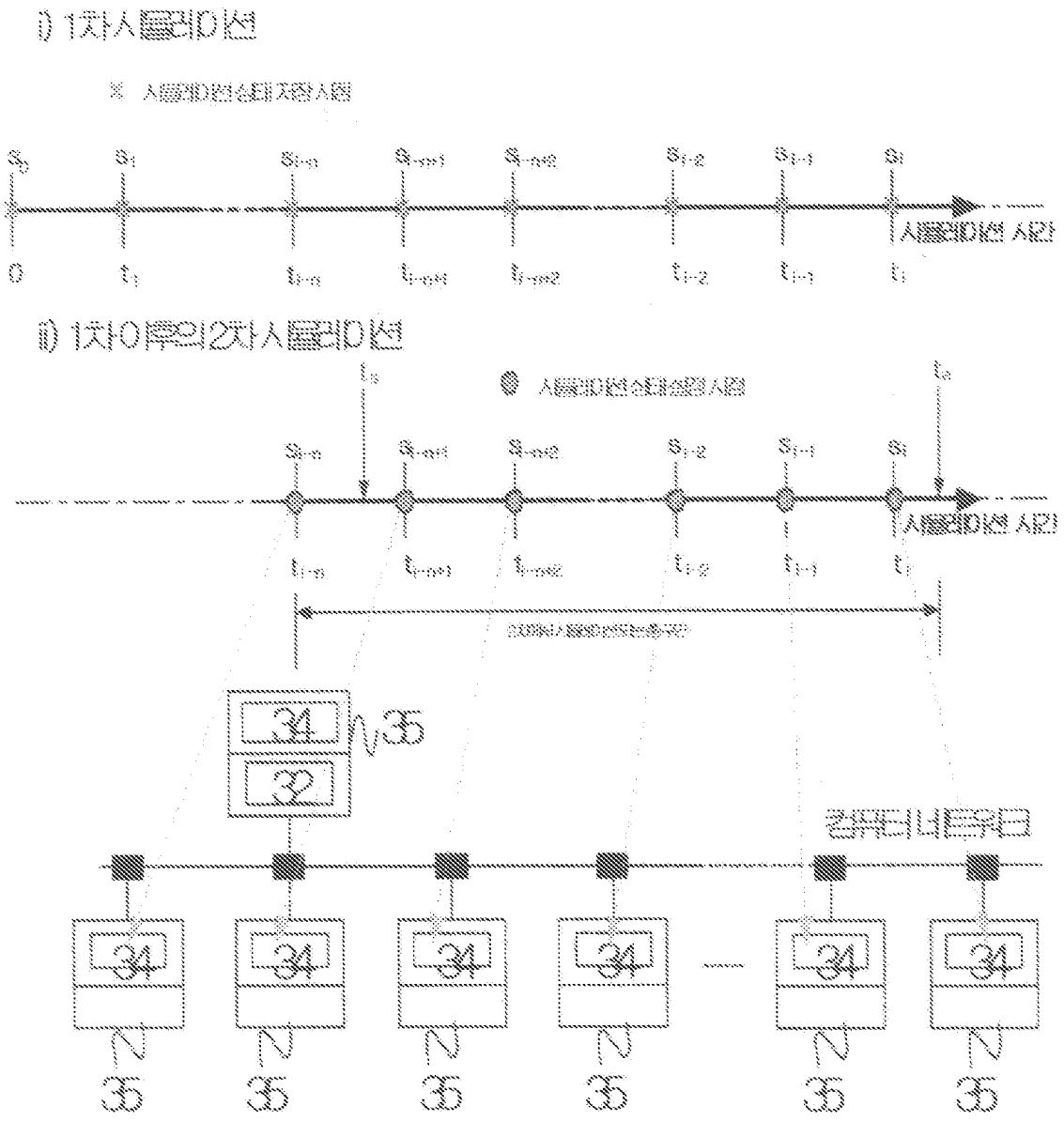
【도 5a】


설계 코드
1차 시뮬레이션 수행 도중에 일정 간격이나 특정 시점에서 시뮬레이션의 상태가 자동적으로 저장되도록 설계 코드에 추가 코드를 부가
100
시뮬레이션의 상태 저장 간격이나 저장 시점 설정
102
1차 시뮬레이션 수행과 동시에 시뮬레이션 상태 정보를 저장
104
탐침이 필요한 시뮬레이션 시간대 (t ~ t) 설정
106
1차 시뮬레이션 과정에서 저장된 시뮬레이션 상태 정보를 이용하여 시뮬레이션을 수행하면서 탐침 실행
108
아니요
설계 코드에 존재하는 설계 오류 발견 및 제거
110
아니요
설계 검증 완료
112
예
종료

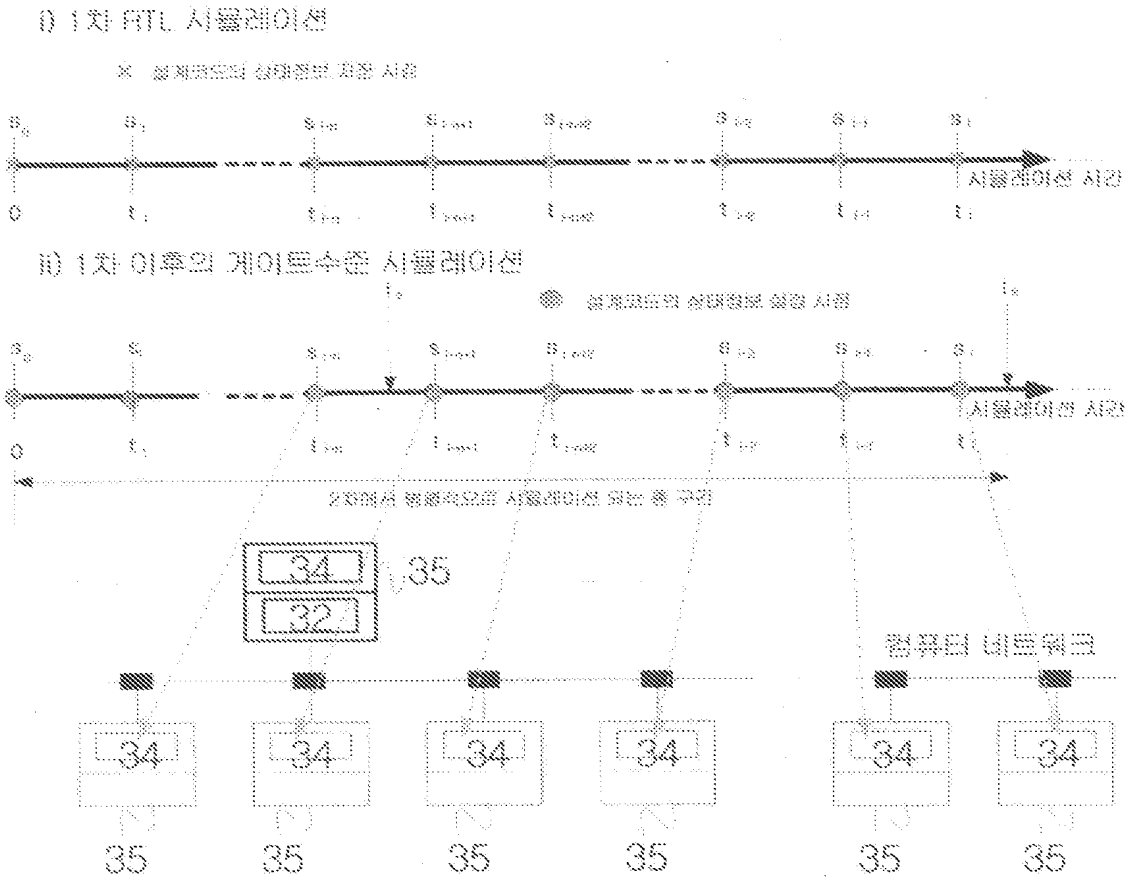
【도 5b】


설계 코드
200
설계 코드에 대한 분할을 수행 2 이상의 설계블럭들을 생성
1차 시뮬레이션 수행 도중에 모든 설계블럭들의 입력과 입출력을 시뮬레이션 전구간에서 탐침하여 저장할 수 있도록 설계 코드에 추가 코드를 추가
201
1차 시뮬레이션 수행과 동시에 각 설계블럭들의 입력과 입출력에 대한 탐침 정보들을 저장한 후, 해당 설계블럭들과 같이 시뮬레이션 컴파일하여 1 이상의 시뮬레이션 실행 파일 생성
204
추가 탐침이 필요한 시그널들이나 변수들 설정
206
추가 탐침이 필요한 시그널들이나 변수들을 포함하는 1 이상의 설계블럭들을 포함하고 있는 1 이상의 시뮬레이션 이진 파일들을 이용한 시뮬레이션 수행하며 해당 시그널들이나 변수들에 대한 탐침 수행
208
아니요
설계 코드에 존재하는 설계 오류 발견 및 제거
210
아니요
설계 검증 완료
212
예
종 료

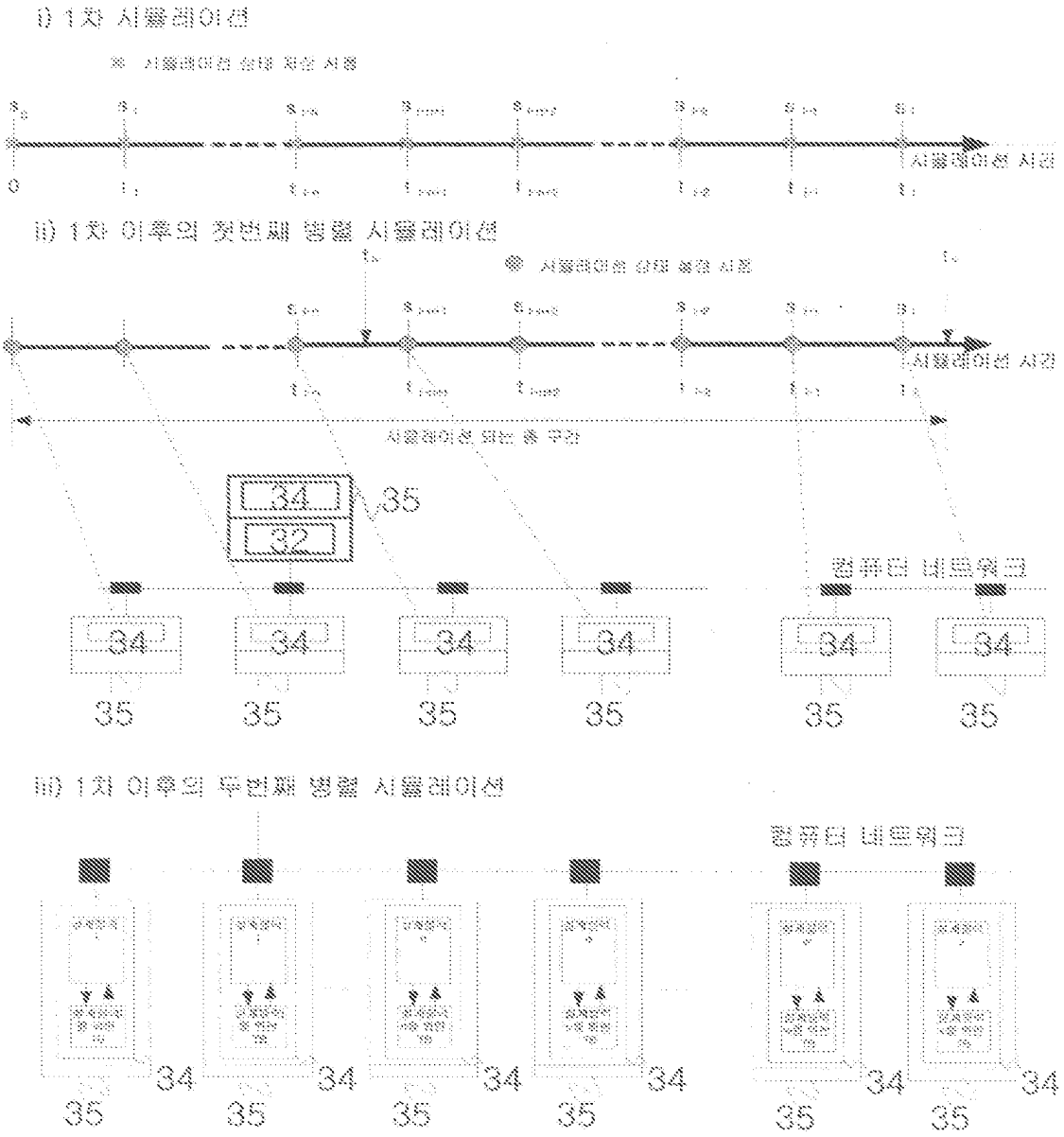
【도 6a】



【도 6b】


i) 1차 RTL 시뮬레이션
시뮬레이션 시간
ii) 1차 이후의 게이트수준 시뮬레이션
시뮬레이션 시간
34
32
35
컴퓨터 네트워크
34
35

【도 6c】


i) 1차 시뮬레이션
시뮬레이션 시간
ii) 1차 이후의 첫번째 병렬 시뮬레이션
시뮬레이션 시간
34
32
35
컴퓨터 네트워크
34
35
iii) 1차 이후의 두번째 병렬 시뮬레이션
컴퓨터 네트워크
34
35